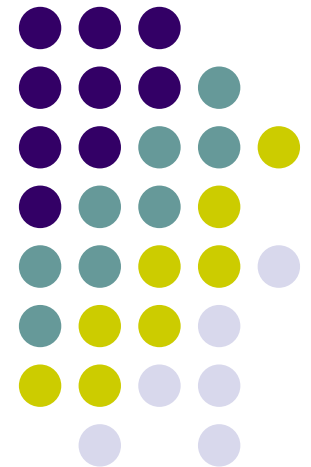


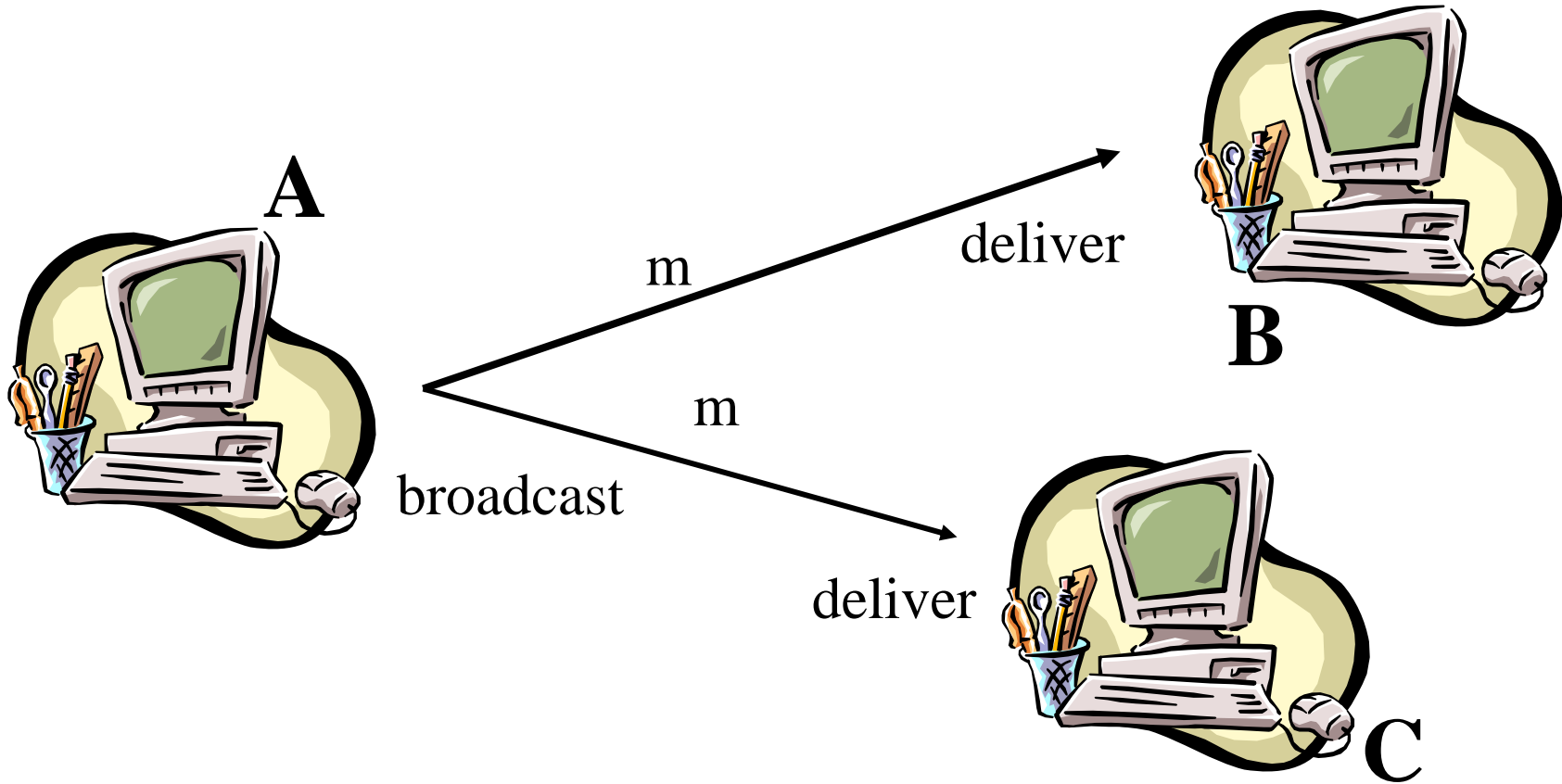
Distributed Algorithms for Building Reliable Systems

Seif Haridi
Reliable Broadcast



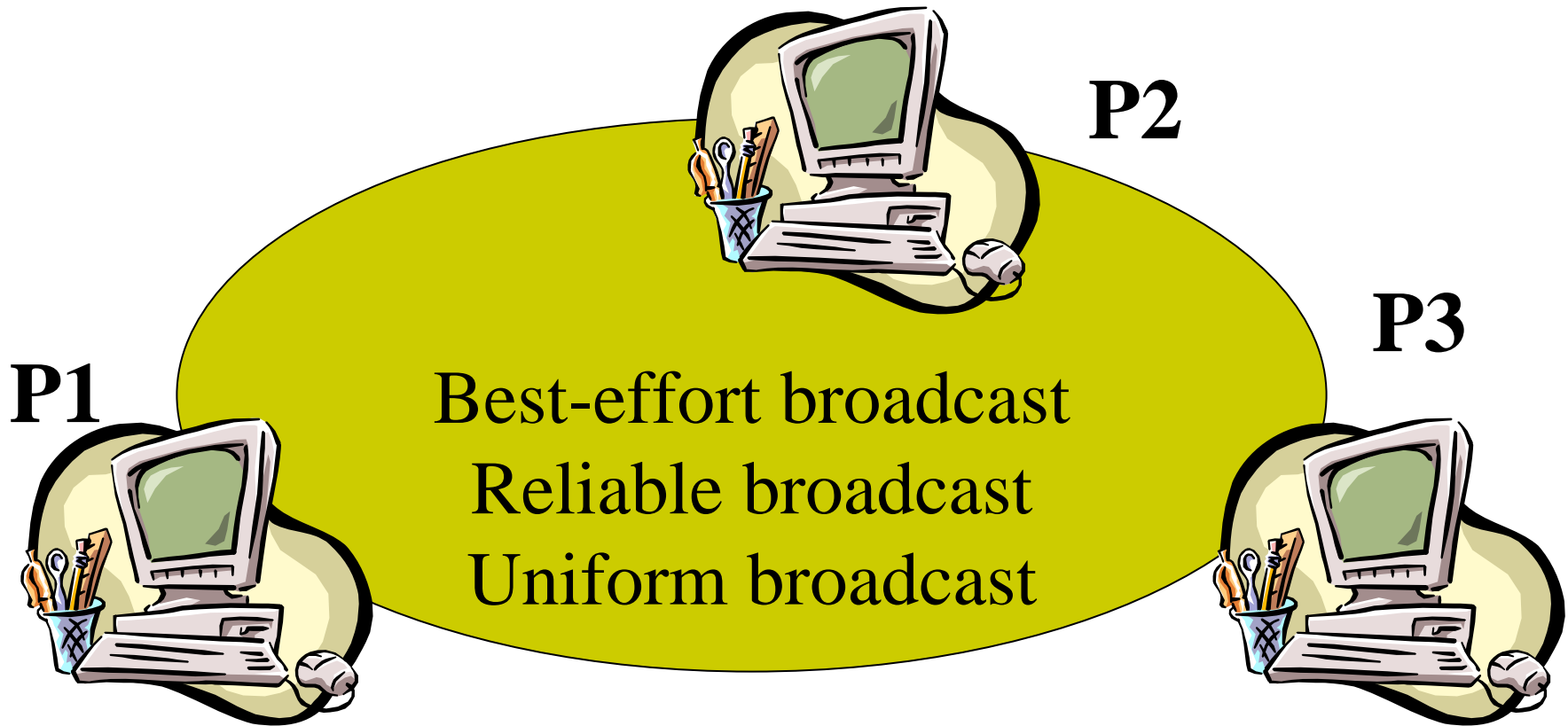


Broadcast



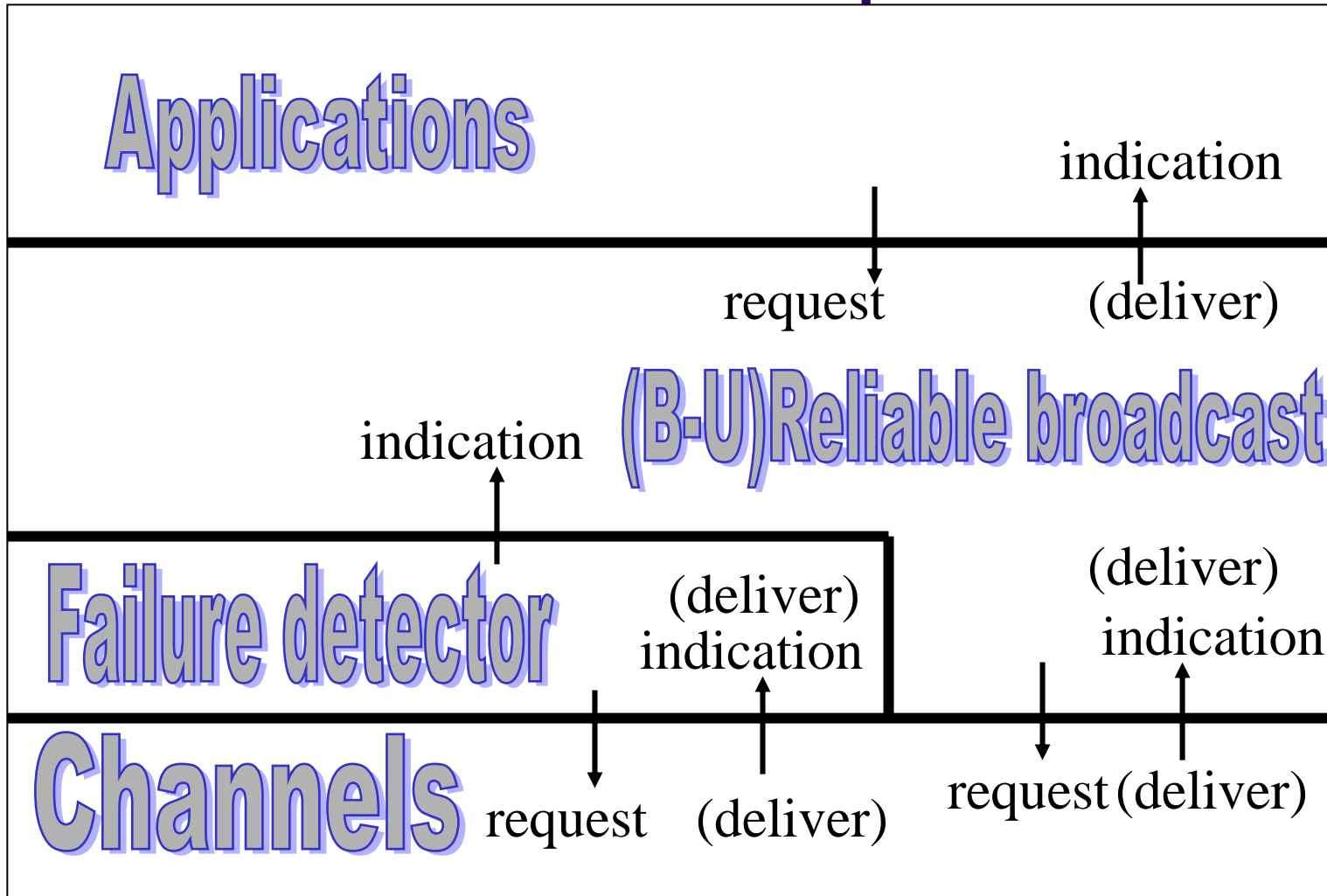


Broadcast abstractions





Modules of a process



Intuition



- Broadcast is useful for instance in applications where some processes subscribe to events published by other processes (e.g., stocks)
- The subscribers might require some **reliability guarantees** from the broadcast service (we say sometimes *quality of service* – *QoS*) that the underlying network does not provide

Overview



- ☛ We shall consider three forms of reliability for a broadcast primitive
- ☛ (1) *Best-effort broadcast*
- ☛ (2) *(Regular) reliable broadcast*
- ☛ (3) *Uniform (reliable) broadcast*
- ☛ We assume a fixed set of processes, from which some may fail
- ☛ We shall give first *specifications* and then *algorithms*



Best-effort broadcast (beb)

• *Events*

• Request: $\langle \text{bebBroadcast } m \rangle$

• Indication: $\langle \text{bebDeliver } \text{src}, m \rangle$

• *Properties: BEB1, BEB2, BEB3*

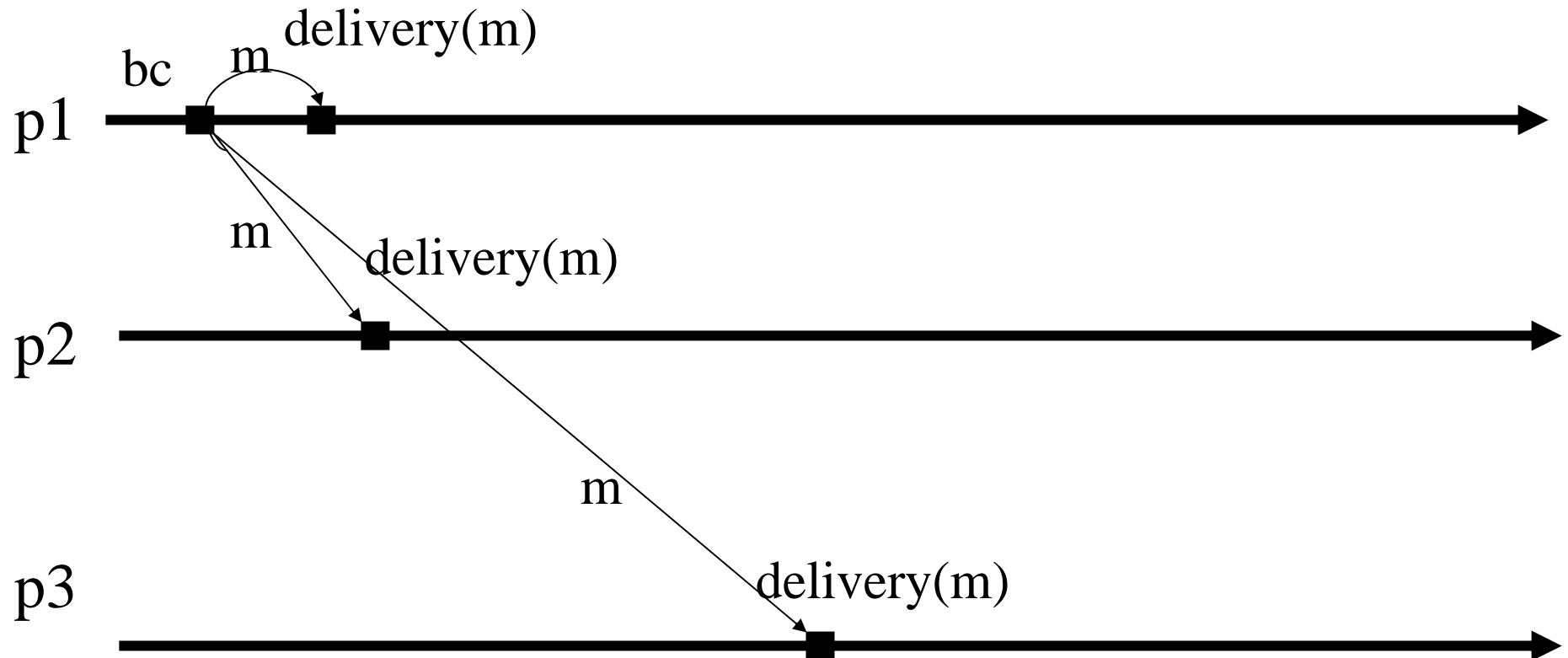


Best-effort broadcast (beb)

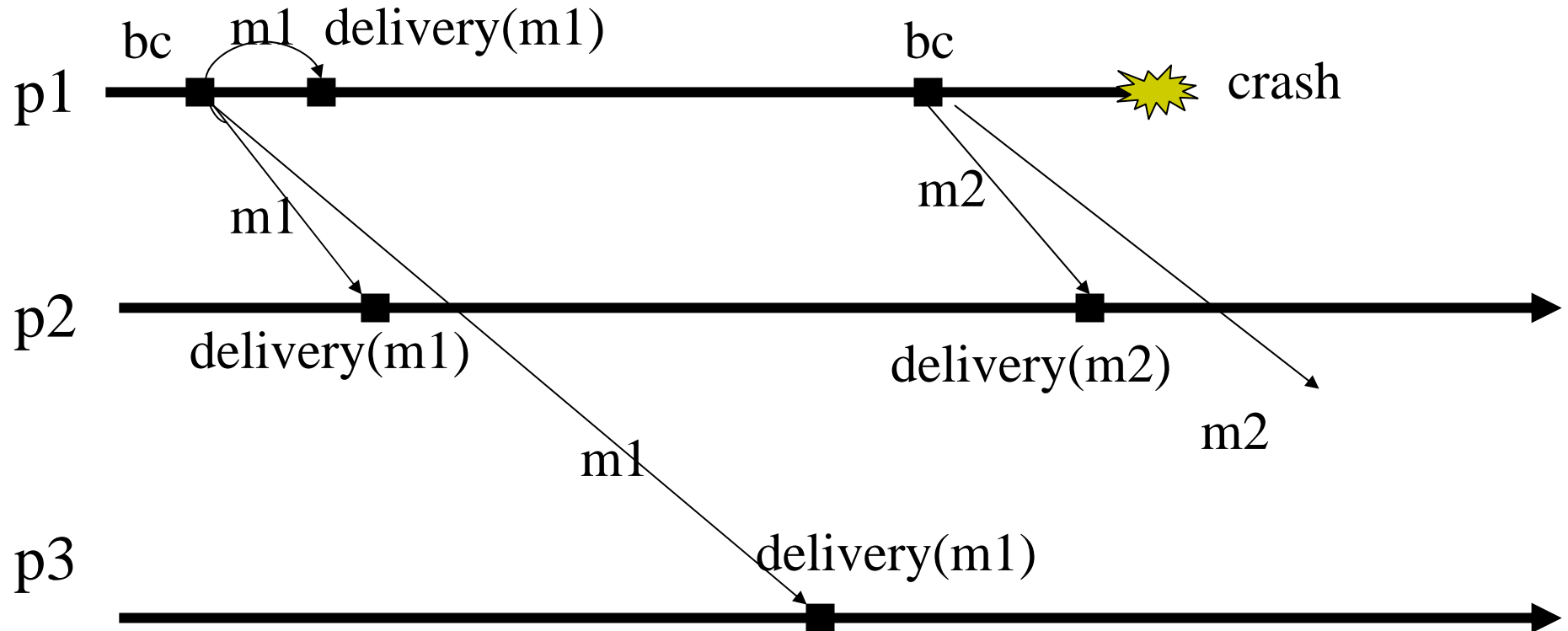
☛ *Properties*

- ☛ ***BEB1. Validity:*** If p_i and p_j are **correct**, then every message broadcast by p_i is eventually delivered by p_j
- ☛ ***BEB2. No duplication:*** No message is delivered more than once
- ☛ ***BEB3. No creation:*** No message is delivered unless it was broadcast

Best-effort broadcast



Best-effort broadcast





Reliable broadcast (rb)

• *Events*

- Request: $\langle \text{rbBroadcast } m \rangle$
- Indication: $\langle \text{rbDeliver src, } m \rangle$

- *Properties: RB1, RB2, RB3, RB4*

Reliable broadcast (rb)



☛ *Properties*

☛ ***RB1 = BEB1.***

☛ ***RB2 = BEB2.***

☛ ***RB3 = BEB3.***

☛ ***RB4. Agreement:*** For any message m , if a **correct process delivers** m , then every correct process delivers m

Reliable broadcast (rb)

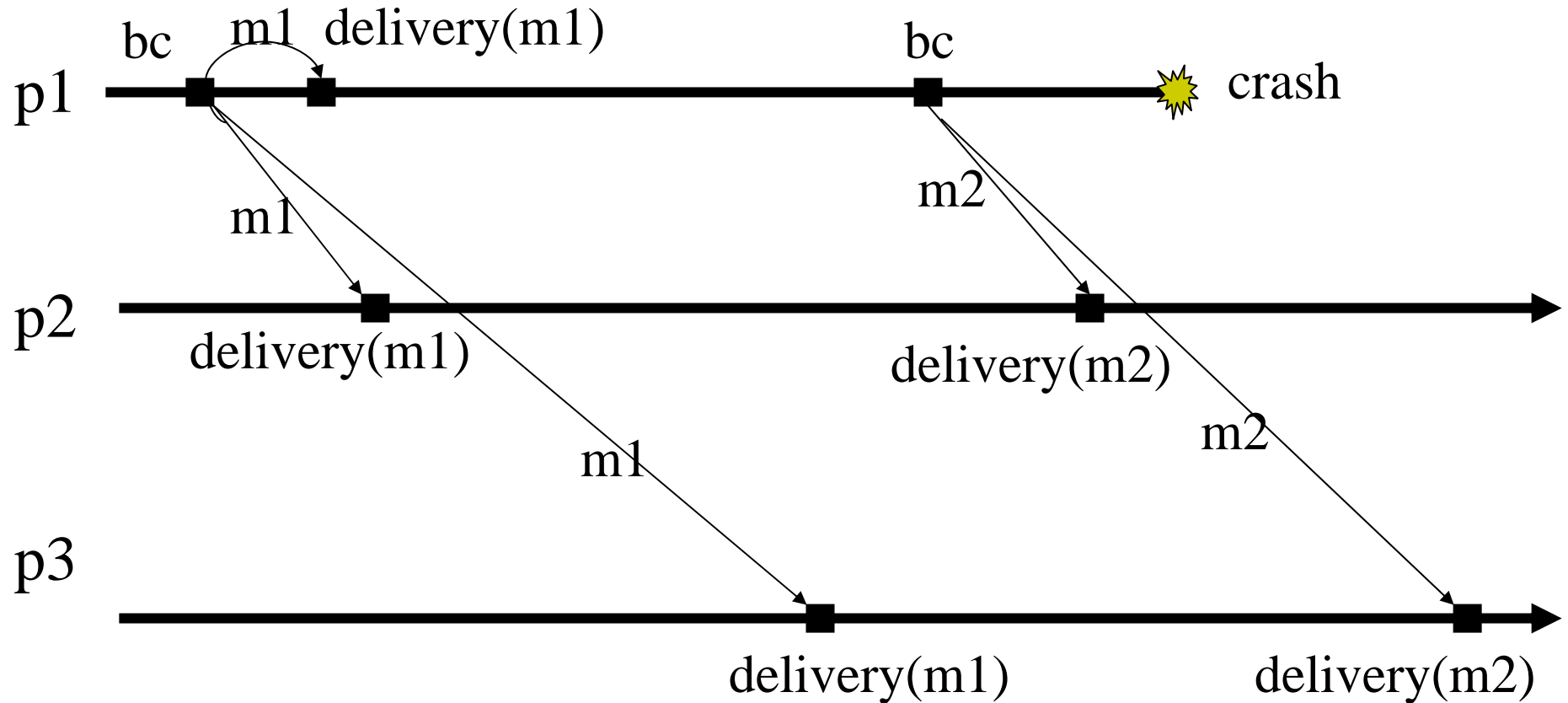


☛ *Properties*

- ☛ ***RB1. Validity.*** If a correct process p_i broadcasts a message m , then p_i eventually delivers m .
- ☛ ***RB2 = BEB2.***
- ☛ ***RB3 = BEB3.***
- ☛ ***RB4. Agreement:*** For any message m , if a **correct process delivers** m , then every correct process delivers m .

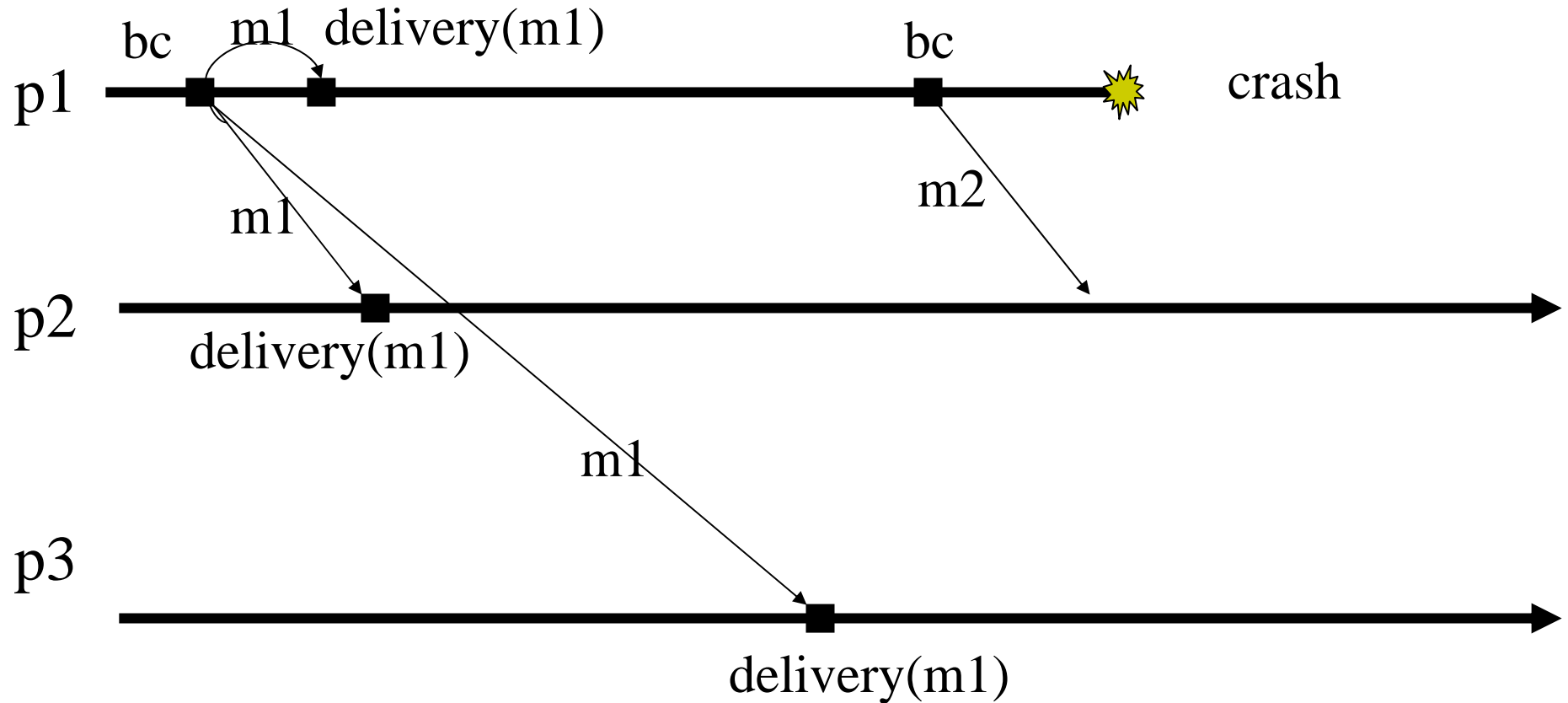
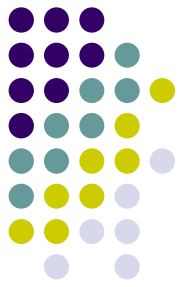
Reliable broadcast

RB4. Agreement



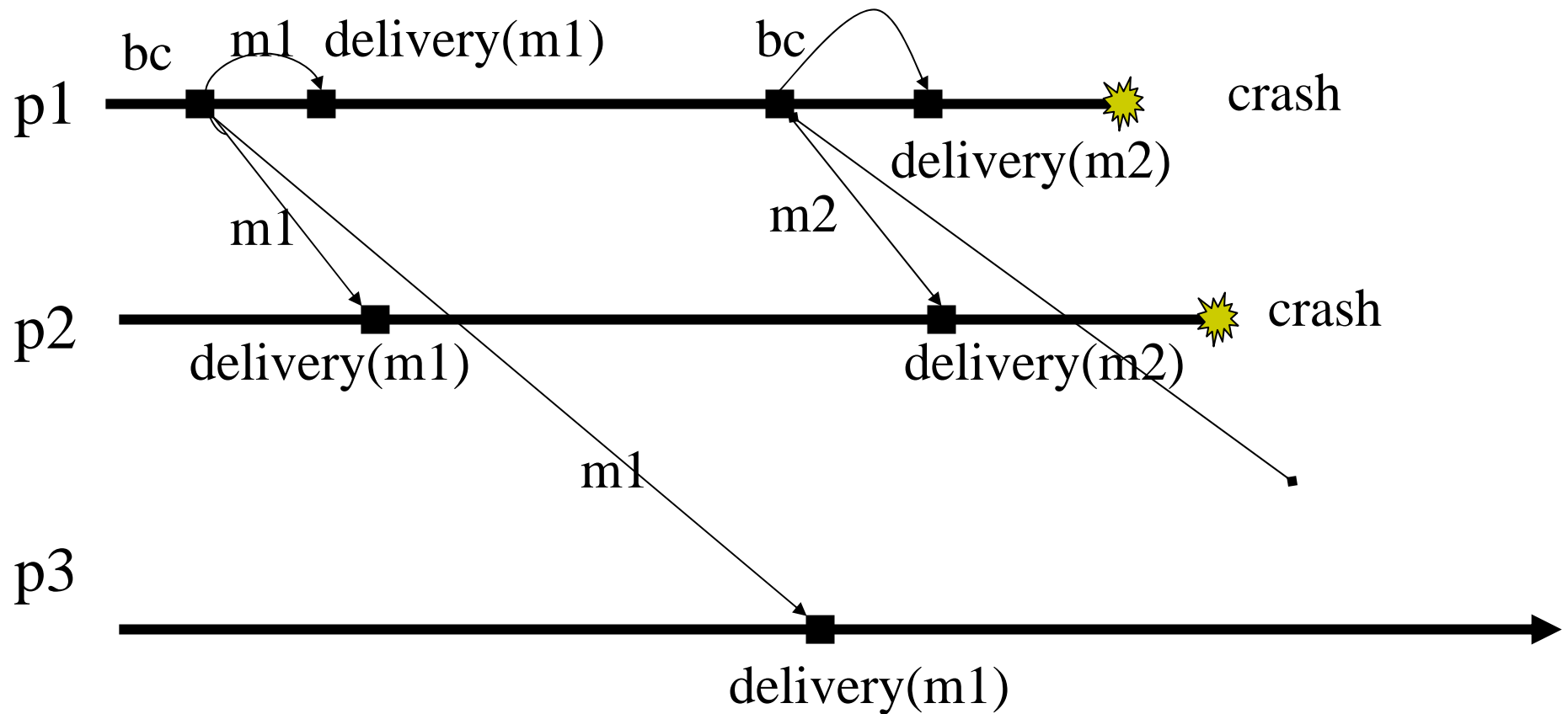
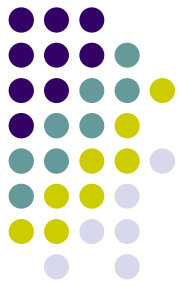
Reliable broadcast

RB4. Agreement



Reliable broadcast

RB4. Agreement





Uniform broadcast (urb)

☛ *Events*

- ☛ Request: $\langle \text{urbBroadcast } m \rangle$
- ☛ Indication: $\langle \text{urbDeliver src, } m \rangle$

- *Properties: URB1, URB2, URB3, URB4*



Uniform broadcast (urb)

☛ *Properties*

☛ ***URB1 = BEB1.***

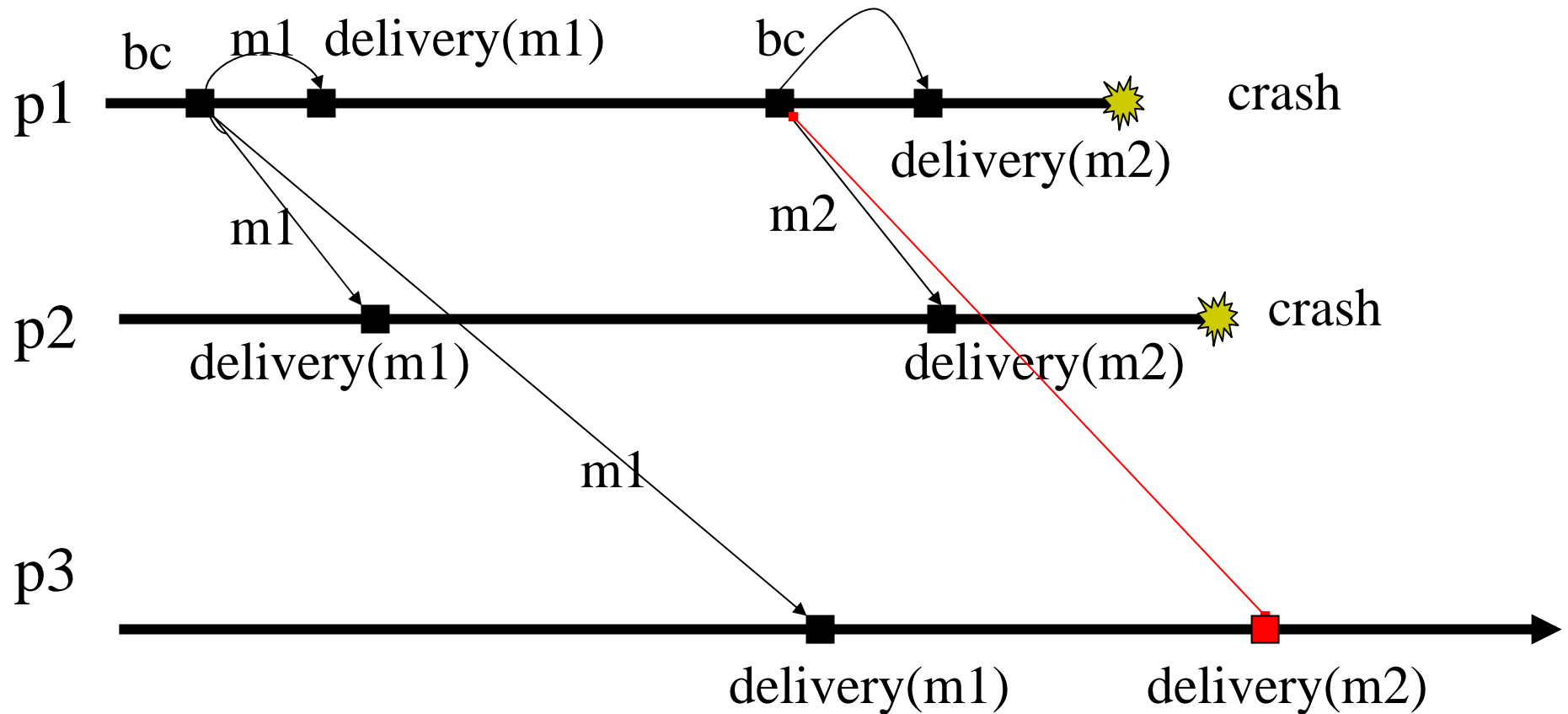
☛ ***URB2 = BEB2.***

☛ ***URB3 = BEB3.***

☛ ***URB4. Uniform Agreement:*** For any message m , if a **process delivers** m , then every correct process delivers m

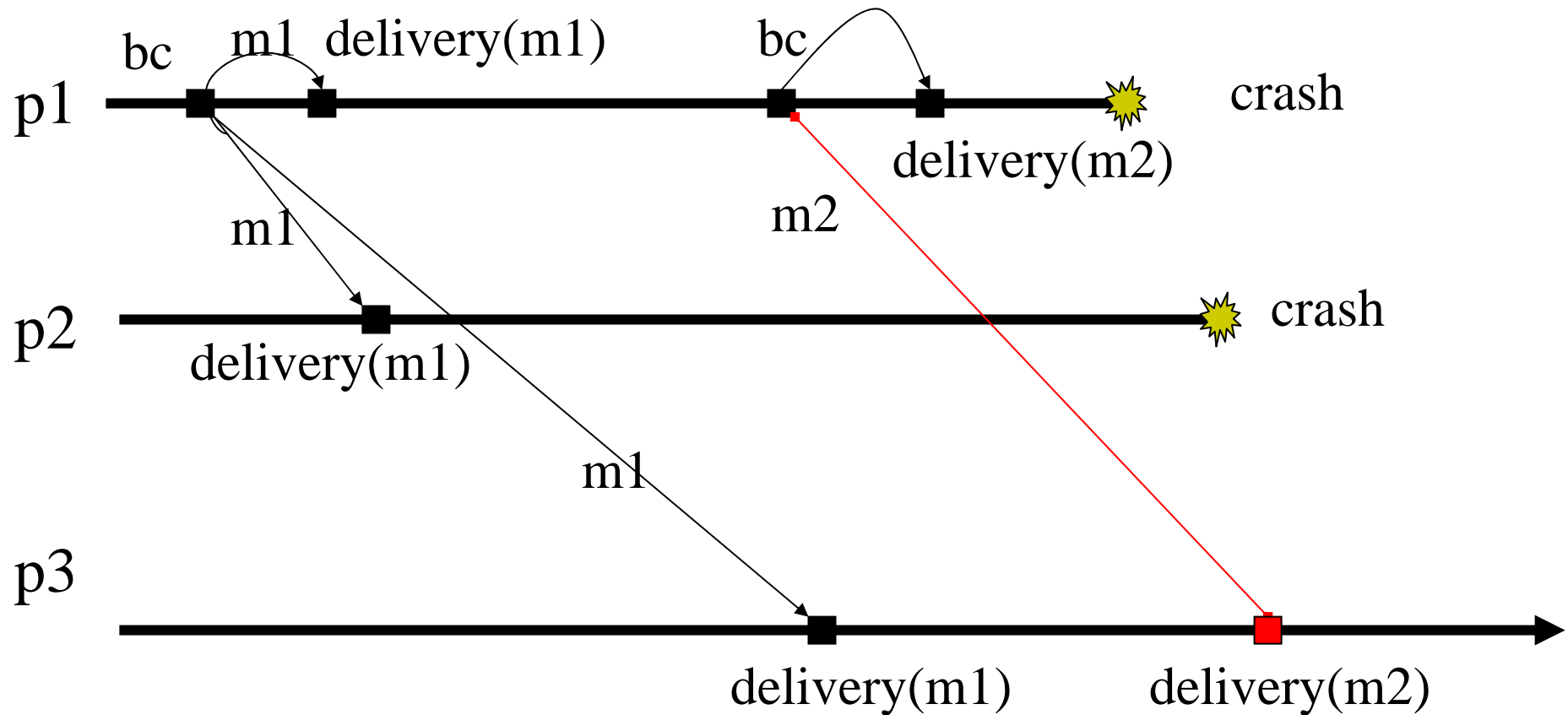
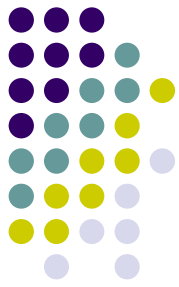
Uniform Reliable broadcast

URB4. Uniform Agreement



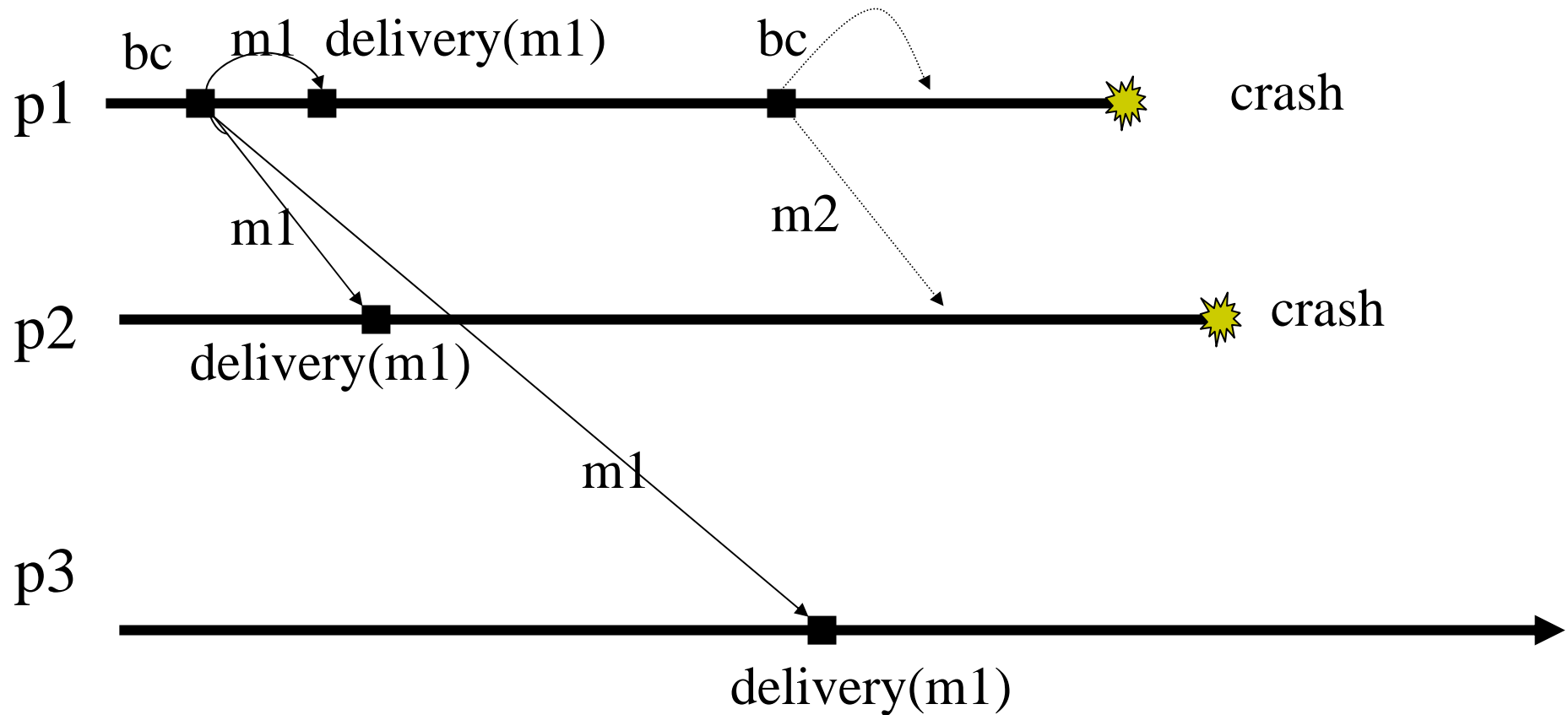
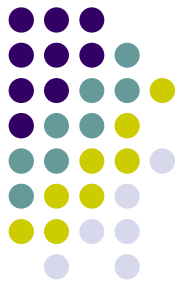
Uniform Reliable broadcast

URB4. Uniform Agreement



Uniform Reliable broadcast

URB4. Uniform Agreement



Overview



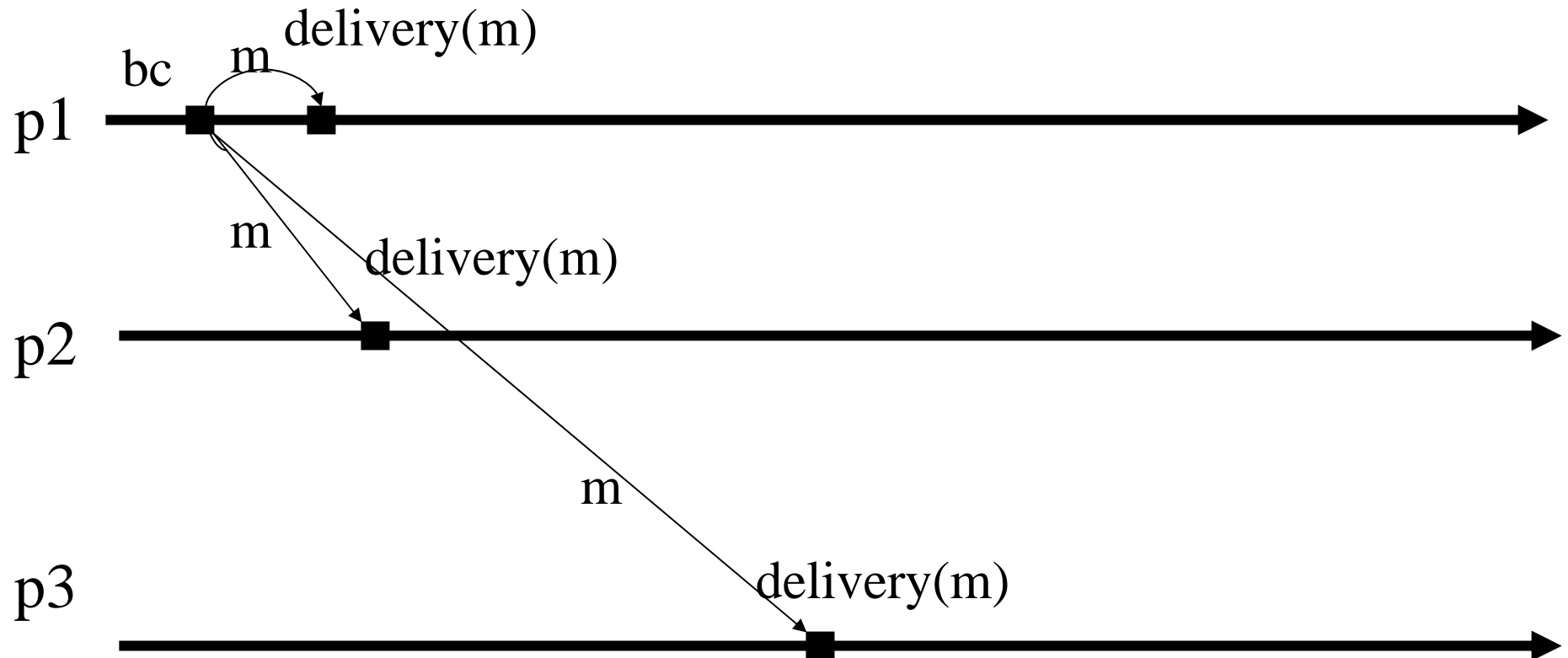
- ☛ We consider three forms of reliability for a broadcast primitive
- ☛ (1) *Best-effort broadcast*
- ☛ (2) *(Regular) reliable broadcast*
- ☛ (3) *Uniform (reliable) broadcast*
- ☛ We give first *specifications* and then *algorithms*



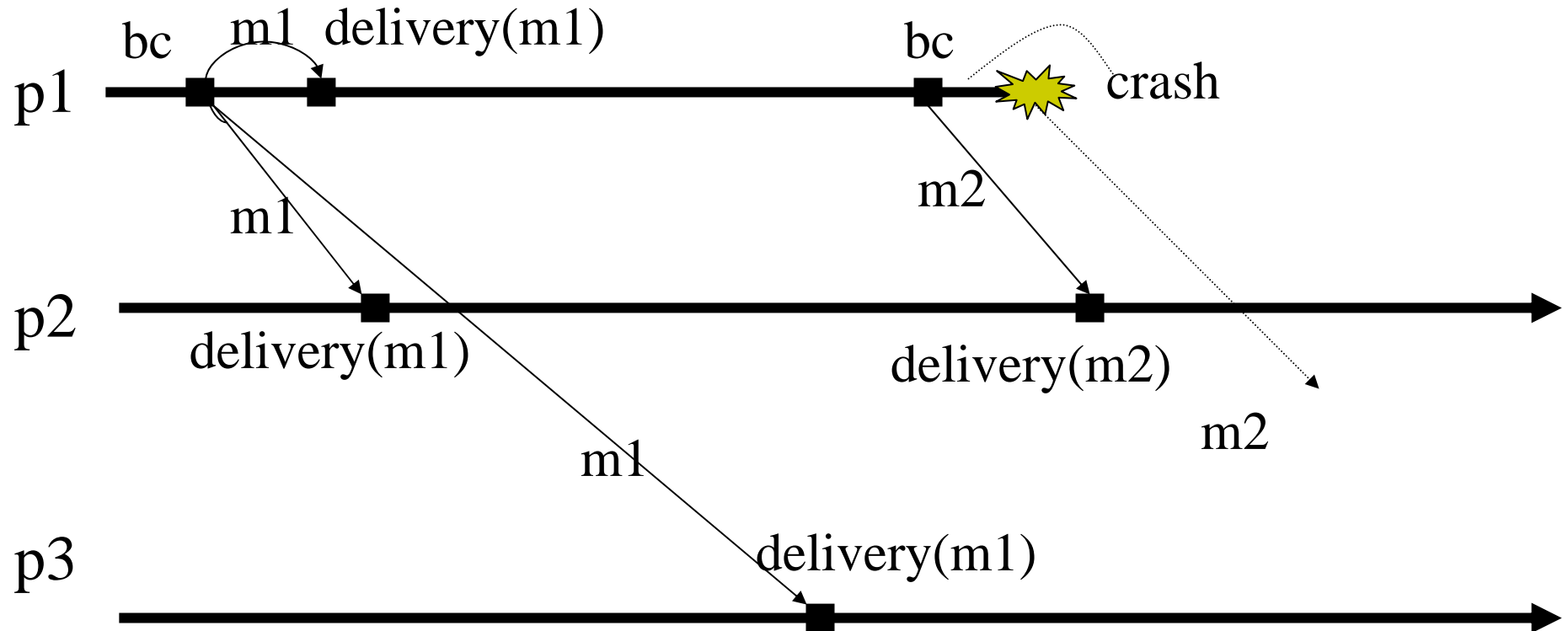
Algorithm (beb)

- ☞ **Implements:** BestEffortBroadcast (beb)
- ☞ **Uses:** PerfectLinks (pp2p)
- ☞ **upon event** $\langle \text{bebBroadcast } m \rangle$ **do**
 - ☞ **forall** $p_i \in \Pi$ **do**
 - ☞ **trigger** $\langle \text{pp2pSend } p_i, m \rangle$
- ☞ **upon event** $\langle \text{pp2pDeliver } p_i, m \rangle$ **do**
 - ☞ **trigger** $\langle \text{bebDeliver } p_i, m \rangle$

Best-effort broadcast



Best-effort broadcast



Algorithm (beb)



☛ *Proof (sketch)*

- ☛ ***BEB1. Validity:*** By the validity property of perfect links and the very facts that (1) the sender sends the message to all and (2) every correct process that pp2pDelivers a message bebDelivers it
- ☛ ***BEB2. No duplication:*** By the no duplication property of perfect links
- ☛ ***BEB3. No creation:*** By the no creation property of the perfect links

Fail-Stop Algorithm

Lazy Reliable Broadcast (rb)



- Requires perfect failure detector
- To rb-Broadcast a message m , a process uses beb-Broadcast to disseminate m to all processes
- A process that gets m rb-delivers it immediately
- If a sender s crashes, a process that rb-delivered messages received from s can detect that crash and relays these messages to all correct processes
- Duplicate messages have to be filtered before rb-delivery

Algorithm (rb)

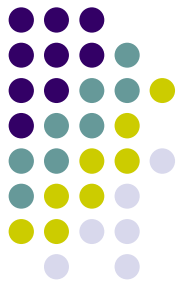


- ☞ **Implements:** ReliableBroadcast (rb)
- ☞ **Uses:**
 - ☞ BestEffortBroadcast (beb)
 - ☞ PerfectFailureDetector (P)
- ☞ **upon event** $\langle \text{Init} \rangle$ **do**
 - ☞ $\text{delivered} := \emptyset$
 - ☞ $\text{correct} := \Pi$
 - ☞ **forall** $p_i \in \Pi$ **do** $\text{from}[p_i] := \emptyset$

Algorithm (rb)



- ☛ **upon event** $\langle \text{rbBroadcast } m \rangle$ **do**
 - ☛ **trigger** $\langle \text{bebBroadcast (DATA, self, } m) \rangle$
- ☛ **upon event** $\langle \text{crash } p_i \rangle$ **do**
 - ☛ $\text{correct} := \text{correct} \setminus \{p_i\}$
 - ☛ **forall** $(s_m, m) \in \text{from}[p_i]$ **do**
 - ☛ **trigger** $\langle \text{bebBroadcast (DATA, } s_m, m) \rangle$
 - ☛ $\text{from}[p_i] := \emptyset$

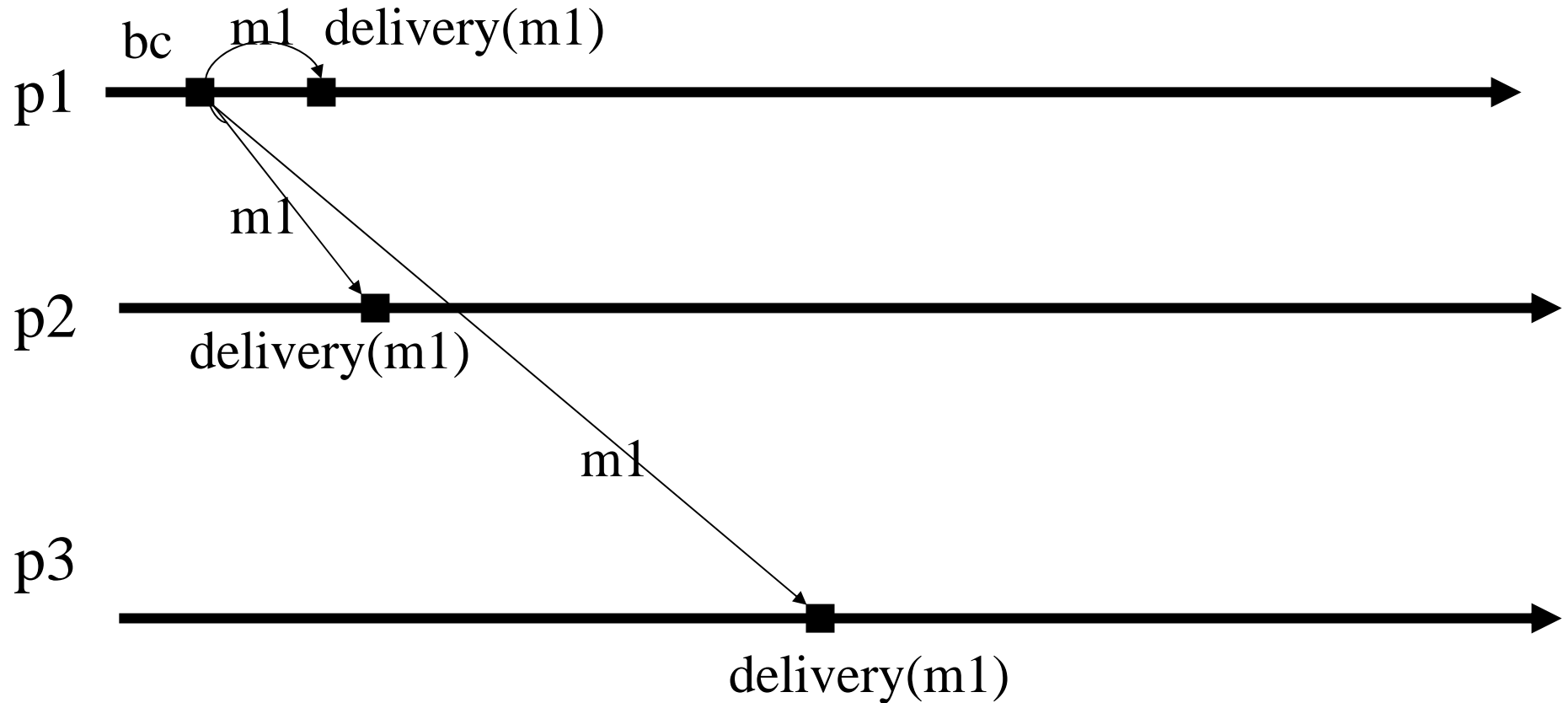
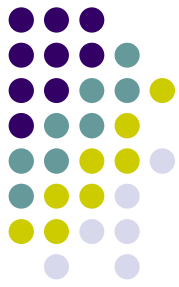


Algorithm (rb – cont'd)

- ☛ **upon event** $\langle \text{bebDeliver } p_i, (\text{DATA}, s_m, m) \rangle$ **do**
 - ☛ **if** $m \notin \text{delivered}$ **then**
 - ☛ $\text{delivered} := \text{delivered} \cup \{m\}$
 - ☛ **trigger** $\langle \text{rbDeliver } s_m, m \rangle$
 - ☛ **if** $p_i \notin \text{correct}$ **then**
 - ☛ **trigger** $\langle \text{bebBroadcast } (\text{DATA}, s_m, m) \rangle$
 - ☛ **else**
 - ☛ $\text{from}[p_i] := \text{from}[p_i] \cup \{ (s_m, m) \}$

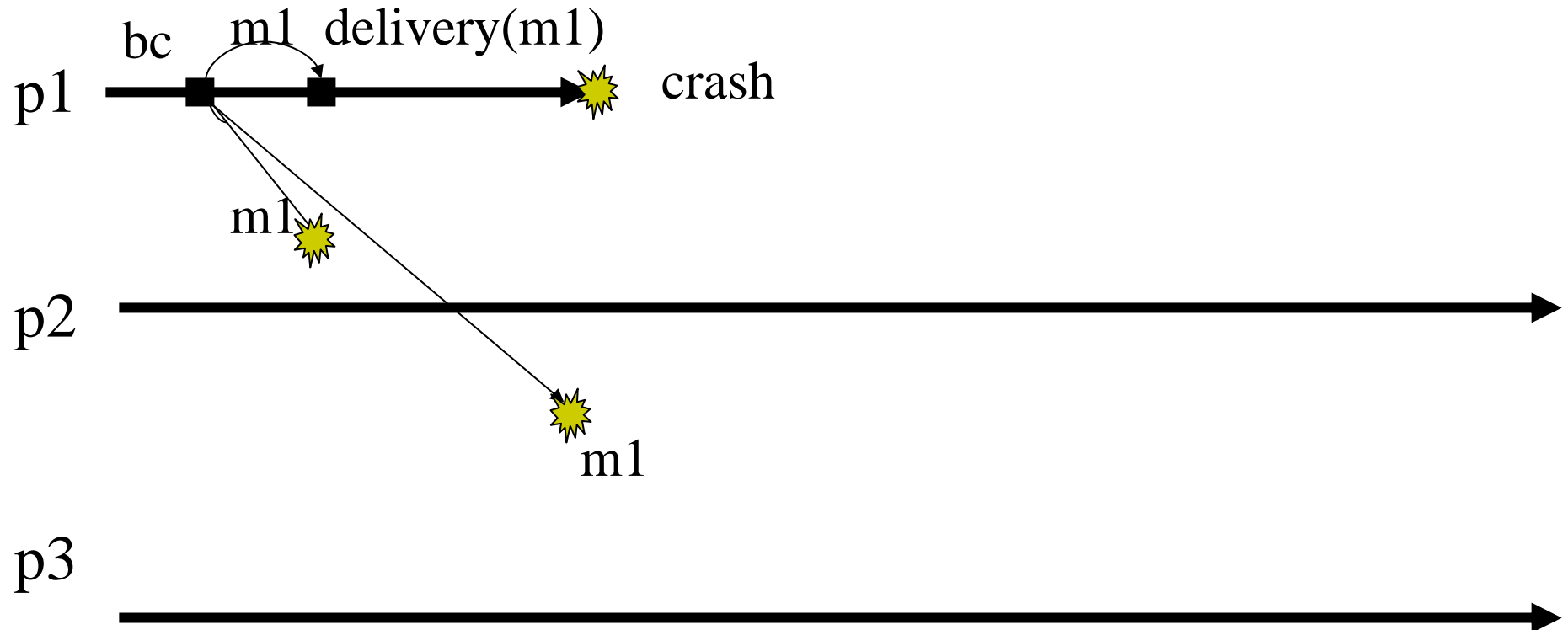
Reliable broadcast

RB4. Agreement: No Crash



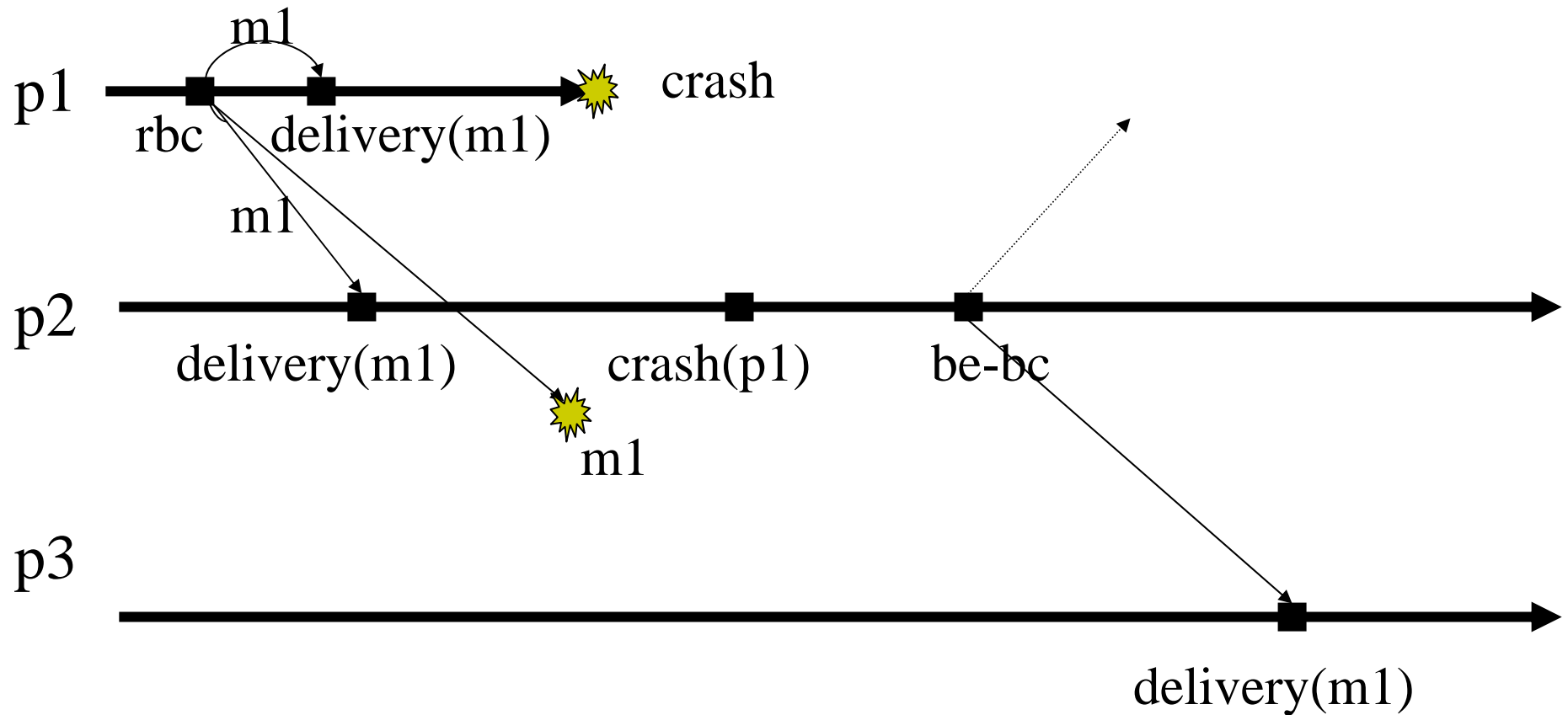
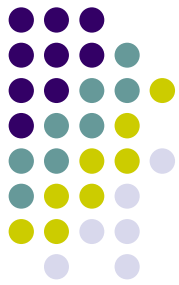
Reliable broadcast

RB4. Agreement: Not uniform



Reliable broadcast

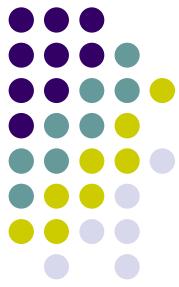
RB4. Agreement: Crash



Algorithm (rb)

Proof (sketch)

- ☞ **RB1. RB2. RB3:** as for the 1st algorithm
- ☞ **RB4. Agreement:** Assume some correct process p_i rb-Delivers a message m rb-Broadcast by some process p_k .
 - ☞ If p_k is correct, then by property BEB1, all correct processes beb-Deliver and then rb-Deliver m .
 - ☞ If p_k crashes, then by the completeness property of P , p_i detects the crash and beb-Broadcasts m to all. Since p_i is correct, then by property BEB1, all correct processes beb-Deliver and then reb-Deliver m .



Algorithm (rb)



➤ *Does the algorithm work for $\diamond P$?*



Fail-Stop All-Ack urb

- In rb uniform agreement is not ensured
 - A process may rb-Deliver and then crash
 - Even if a process beb-Broadcast before crashing the messages might not reach their destinations (WHY)
- A process urb-Delivers m only when it knows that m has been beb-delivered by all correct processes

Algorithm (urb) Fail-stop all ack



- ☞ **Implements:** uniformBroadcast (urb)

- ☞ **Uses:**

 - ☞ BestEffortBroadcast (beb)

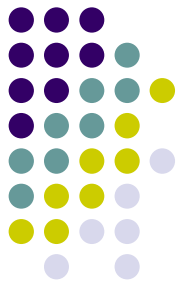
 - ☞ PerfectFailureDetector (P)

- ☞ **upon event** $\langle \text{Init} \rangle$ **do**

 - ☞ $\text{correct} := \Pi$

 - ☞ $\text{delivered} := \text{pending} := \emptyset$

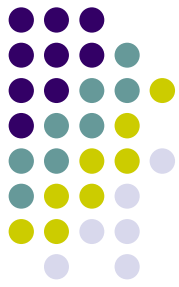
 - ☞ **forall** m **do** $\text{ack}[m] := \emptyset$



Algorithm (urb – cont'd)

- **function** canDeliver(m) **returns** boolean **is**
 - **return** correct \subseteq ack[m]
- **upon event** \langle crash pi \rangle **do**
 - correct := correct \setminus {pi}
- **upon event** \langle urbBroadcast m \rangle **do**
 - pending := pending \cup { (self, m) }
 - **trigger** \langle bebBroadcast (DATA, self, m) \rangle

Algorithm (urb – cont'd)

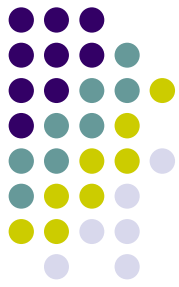


- ☛ **upon event** $\langle \text{bebDeliver } pi, (\text{DATA}, s_m, m) \rangle$ **do**
 - ☛ $\text{ack}[m] := \text{ack}[m] \cup \{pi\}$
 - ☛ **if** $(s_m, m) \notin \text{pending}$ **then**
 - ☛ $\text{pending} := \text{pending} \cup (s_m, m)$
 - ☛ **trigger** $\langle \text{bebBroadcast } (\text{DATA } s_m, m) \rangle$

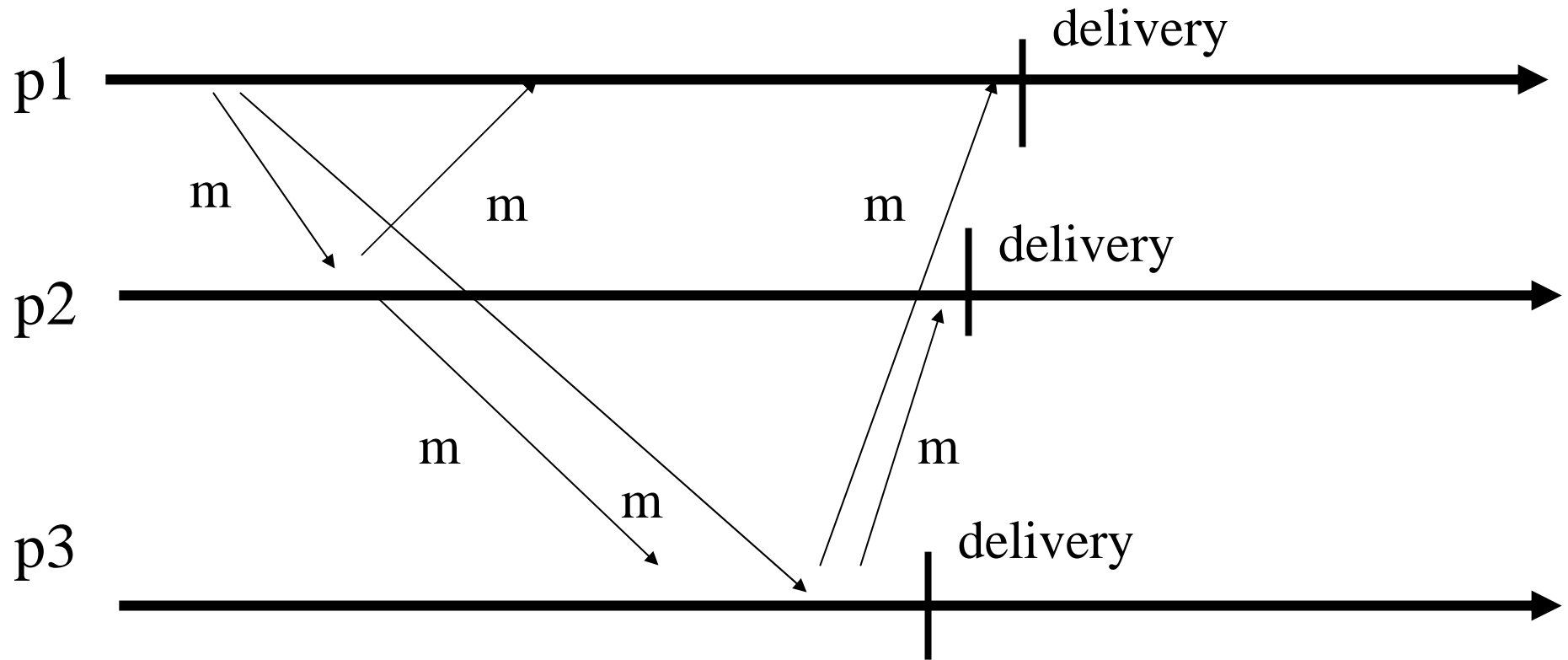


Algorithm (urb – cont'd)

- **Upon exists** $(s_m, m) \in \text{pending}$ **such that** $\text{canDeliver}(m)$ and $m \notin \text{delivered}$ **do**
 - $\text{delivered} := \text{delivered} \cup \{m\}$
 - **trigger** $\langle \text{urbDeliver } s_m, m \rangle$

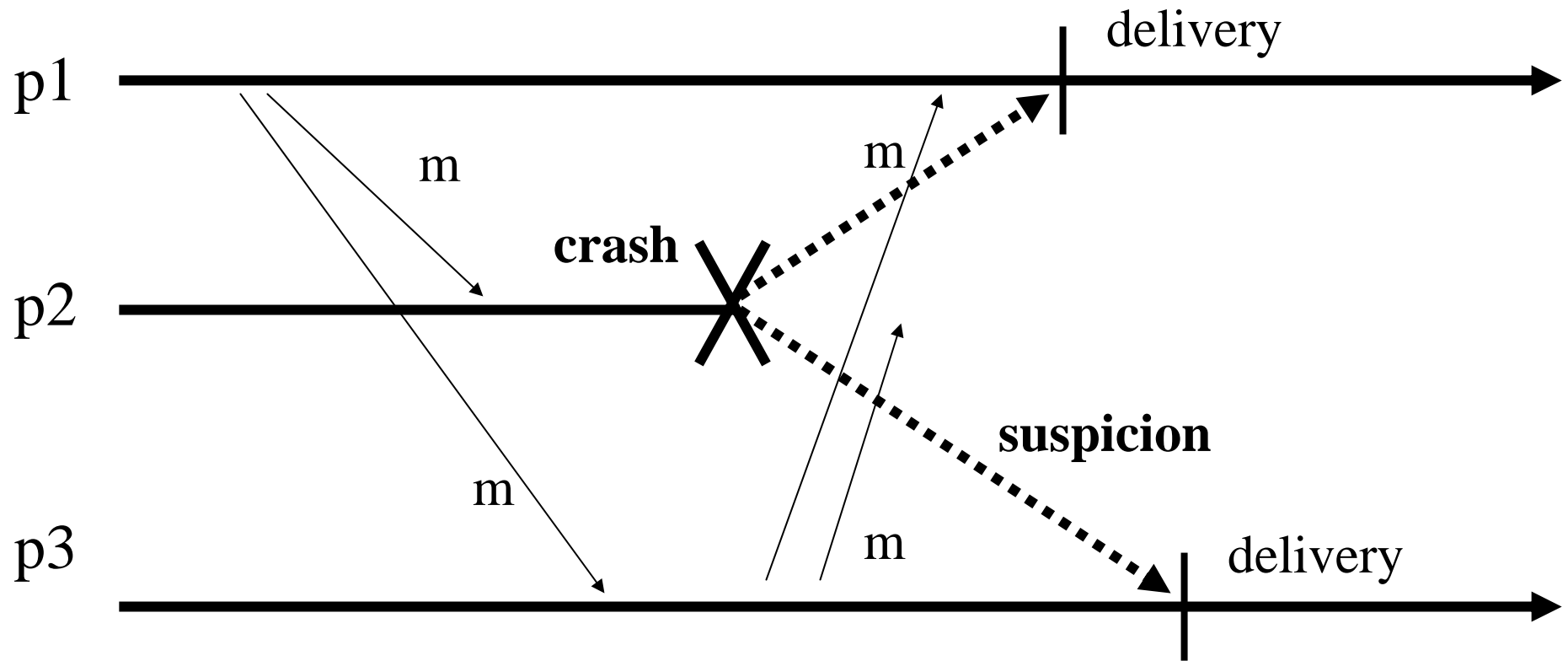


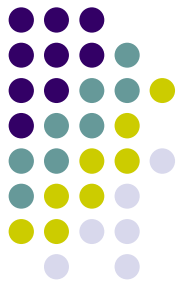
Algorithm (urb)





Algorithm (urb)



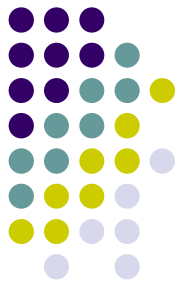


Algorithm (urb)

☛ *Proof (sketch)*

- ☛ **URB2. URB3:** follow from BEB2 and BEB3
- ☛ **A simple lemma:** *If a correct process p_i bebDelivers a message m , then p_i eventually urbDelivers m .*
- ☛ Any process that bebDelivers m bebBroadcasts m . By the completeness property of the failure detector and property BEB1, there is a time at which p_i bebDelivers m from every correct process and hence urbDelivers m .

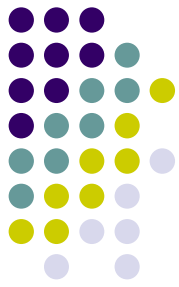
Algorithm (urb)



☛ *Proof (sketch)*

- ☛ **URB1. Validity:** If a correct process p_i urb-Broadcasts a message m , then p_i eventually beb-Broadcasts and bebDelivers m : by our lemma, p_i urbDelivers m .
- ☛ **URB4. Agreement:** Assume some process p_i urb-Delivers a message m . By the algorithm and the completeness and accuracy properties of the failure detector, every correct process beb-Delivers m . By our lemma, every correct process will urb-Deliver m .

Algorithm (urb)



➤ *Does the algorithm work for $\diamond P$?*



Fail-Silent Majority-Ack urb

- No failure detector
- Requires Majority to be correct
- Each process waits to `urbDeliver` a message until a majority of processes have seen the message



Fail-Silent Majority-Ack urb

Algorithm 3.5 Majority-Ack Uniform Reliable Broadcast

Implements:

UniformReliableBroadcast (urb).

Extends:

All-Ack Uniform Reliable Broadcast (Algorithm 3.4).

Uses:

BestEffortBroadcast (beb).

function canDeliver(m) returns boolean is

return ($|\text{ack}_m| > N/2$)

// Except for the function above, and the non-use of the

// perfect failure detector, same as Algorithm 3.4.
